Paradigmas de lenguajes de programación

Eduardo Bonelli

Departamento de Computación
FCEyN
UBA

"A language that doesn't affect the way you think about programming, is not worth knowing"

Epigrams in Programming, Alan Perlis (primer ganador del Turing Award), ACM SIGPLAN, Sept. 1982

20 de marzo de 2007

Objetivos del curso

Conocer los pilares conceptuales sobre los cuales se erigen los lenguajes de programación de modo de poder

- Comparar lenguajes
- Seleccionar el más adecuado para una determinada tarea
- Prepararse para lenguajes/paradigmas futuros

Enfoque del curso

1. Angulo conceptual
   - Introducción informal de conceptos a través de ejemplos.
2. Angulo aplicado
   - Implementación de los conceptos a través de intérpretes.
3. Angulo de fundamentos
   - Introducir las bases rigurosas (lógicas y matemáticas) sobre las que se sustentan cada uno de los paradigmas o parte de los mismos

Enfoque del curso

1. Angulo conceptual
   - Introducción informal de conceptos a través de ejemplos.
2. Angulo aplicado
   - Implementación de los conceptos a través de intérpretes.
3. Angulo de fundamentos
   - Introducir las bases rigurosas (lógicas y matemáticas) sobre las que se sustentan cada uno de los paradigmas o parte de los mismos

Enfoque del curso

1. Angulo conceptual
   - Introducción informal de conceptos a través de ejemplos.
2. Angulo aplicado
   - Implementación de los conceptos a través de intérpretes.
3. Angulo de fundamentos
   - Introducir las bases rigurosas (lógicas y matemáticas) sobre las que se sustentan cada uno de los paradigmas o parte de los mismos

Algunos temas cubiertos - Angulo conceptual

- Inducción (variantes) y recursión
- binding estático/dinámico, procedimientos de primera clase, recursión `letrec`, modalidades de pasaje de parámetros
- Asignación y efectos laterales
- Expresiones de tipo, sistema de tipos, checkeo de tipos, inferencia de tipos
- Resolución en lógica proposicional y de primer orden, Cláusulas de Horn, unificación, refutación, resolución SLD
- Objeto, clase, herencia, method dispatch estático/dinámico, polimorfismo, subtipos, sistemas de tipos invariantes

Algunos temas cubiertos - Angulo aplicado

- Sintaxis, intérprete y type-checker para un lenguaje funcional con procedimientos, let-binding, recursión, referencias.
- Sintaxis e intérprete de una versión reducida de Prolog.
- Sintaxis, intérprete y type-checker para un lenguaje orientado a objetos con clases, métodos, herencia simple, polimorfismo de subclase.

Algunos temas cubiertos - Angulo de fundamentos

- Lambda cálculo
  - Sintaxis y ejemplos de programación
  - Sistemas de tipos e inferencia
  - Semántica operacional
- Resolución
  - En lógica proposicional
  - En lógica de primer orden
  - SLD
- Prog. orientada a objetos
  - Sistemas de tipos, herencia como subtipado
  - POO en lambda cálculo con subtipado, registros y referencias

Formato de presentación

- Ángulo conceptual y de fundamentos
  - Clases teóricas y prácticas
  - Prácticas
  - TPs
- Ángulo aplicado (intérpretes)
  - "TPcitos"
  - Sin obligación de entrega

Cátedra y modalidad

- Modalidad: Clases teóricas y prácticas

Teoría

- Eduardo Bonelli (`ebonelli@dc.uba.ar`)
- Martes: 17:30hs a 20:30hs

Práctica

- Ariel Arbiser (JTP), Alejo Capparelli, Pablo Coll (JTP), Laura Lowenthal, Pablo Heiber, Guido de Caso, Mariano Moscato
- Jueves: 19:30hs a 22:30hs
- **NOTA:** El jueves 22 de marzo se dictará teoría

Recursos

Bibliografía

- Textos: no hay un texto principal, se utilizan varios, referencias en página web
- Apuntes: serán introducidos oportunamente
- Papers
- Transparencias de teóricas

Página web

- Información al día del curso, consultar periódicamente

Mailing list

- ¡Hacer todas las preguntas y consultas que quieran!

Recursos

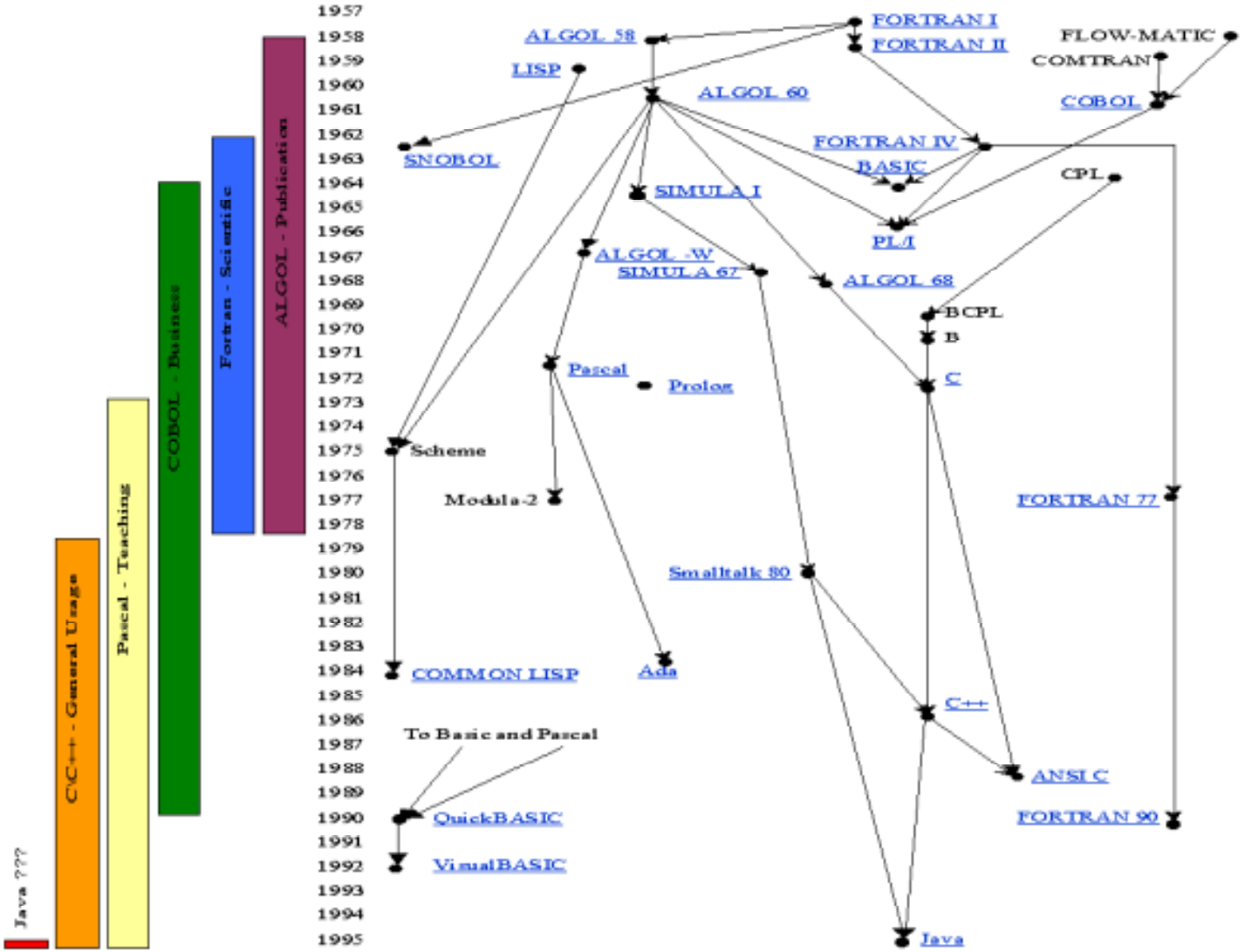
Software

- Implementaremos todos los intérpretes en Haskell
  - El intérprete (Hugs): `http://www.haskell.org`
  - Documentos (accesibles en el mismo sitio):
    - *A Gentle Introduction to Haskell*, Paul Hudak, John Peterson y Joseph H. Fasel.
    - *The Hugs 98 User Manual*, Mark P. Jones y John Peterson, 1999. (Manual del usuario; modestas 84 páginas)
    - *Haskell 98 Language and Libraries - The Revised Report*, Simon Peyton Jones (ed.), 2002. (La referencia definitiva sobre Hugs98, 277 páginas....)
- SWI-Prolog (programación lógica)
- Visualworks (Smalltalk - programación orientada a objetos)

Lenguajes de Programación

Lenguajes de Programación (LP)

- lenguaje usado para comunicar instrucciones a una computadora
- instrucciones describen *cómputos* que llevará a cabo la computadora
- noción de cómputo puede formalizarse (ej. máquinas de Turing, cálculo lambda, funciones recursivas, etc.): conjunto de funciones computables
- *computacionalmente completo* si puede expresar todas las funciones computables


1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
ALGOL - Publication
Fortran - Scientific
COBOL - Business
Pascal - Teaching
C/C++ - General Usage
Java ???
ALGOL 58
FORTRAN I
FORTRAN II
FLOW-MATIC
COMTRAN
LISP
ALGOL 60
COBOL
FORTRAN IV
SNOBOL
BASIC
CPL
SIMULA I
PL/I
ALGOL -W
SIMULA 67
ALGOL 68
BCPL
B
Pascal
Prolog
C
Scheme
Modula-2
FORTRAN 77
Smalltalk 80
COMMON LISP
Ada
C++
To Basic and Pascal
ANSI C
QuickBASIC
FORTRAN 90
VisualBASIC
Java

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

Definición de un Lenguaje

- Sintaxis
  - descripción del conjunto de secuencias de símbolos considerados como programas válidos
  - teoría de lenguajes formales bien desarrollada, comenzando a mediados de 1950 con Noam Chomsky como pionero
  - notación BNF (y EBNF) ampliamente utilizada; desarrollada por John Backus para Algol 58, modificada por Peter Naur para Algol 60
  - amplio abanico de herramientas para generar analizadores léxicos y parsers a partir de notación formal como BNF
- Sistema de Tipos
- Semántica

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos

  - propósito: prevenir errores en tiempo de ejecución
  - en general, requiere anotaciones de tipo en el código fuente
  - ejemplos: evitar sumar booleanos, aplicar función a número incorrecto de argumentos.
  - análisis de tipos en tiempo de compilación: *chequeo de tipos estático*
  - análisis de tipos en tiempo de ejecución: *chequeo de tipos dinámico*
  - veremos en detalle en el Eje de Fundamentos

- Semántica

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

- descripción del *significado* de instrucciones y expresiones
- puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
- ¿Por qué semántica formal?[a]:
  - Destructor británico H.M.S. Sheffield hundido en guerra de Malvinas. El radar de alerta de la nave estaba programado para identificar el misil Exocet como .ªliado"debido a que el arsenal Inglés incluye el "homing device" de los Exocet y permitió que el misil alcanzara su blanco (el H.M.S. Sheffield).

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

- descripción del *significado* de instrucciones y expresiones
- puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
- ¿Por qué semántica formal?
  - Votos perdidos por computadora en Toronto. El distrito de Toronto finalmente abandonó votación electrónica.

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

- descripción del *significado* de instrucciones y expresiones
- puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
- ¿Por qué semántica formal?
  - 225 de los 254 pasajeros de Korean Airlines KAL 901 en Guam fallecen en accidente. Bug descubierto en altímetro barométrico del Ground Proximity Warning System (GPWS).

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

- descripción del *significado* de instrucciones y expresiones
- puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
- ¿Por qué semántica formal?
  - Falla en el despegue del satélite Ariane 5 causado por software error en la rutina de manejo de excepciones resultante de una mala conversión de un punto flotante de 64-bit a un entero.

Sintaxis - Ejemplo (BNF)

Ejemplo

$$\begin{array}{lll} \langle \text{Prog} \rangle & ::= & (\langle \text{Prog} \rangle \texttt{ union } \langle \text{Prog} \rangle) \mid (\langle \text{Elem} \rangle \texttt{ into } \langle \text{Prog} \rangle) \mid \\ & & \texttt{emptyset} \\ \langle \text{Elem} \rangle & ::= & \texttt{a} \mid \texttt{b} \mid \texttt{c} \end{array}$$

Ejemplos de programas válidos

`(a into emptyset)`, `(b into (a into emptyset))`,
`((a into emptyset) union emptyset)`

Ejemplos de programas no válidos

`(a into union)`, `d union`

Semántica - Ejemplo (denotacional)

- Fijar una estructura matemática. Ej. $E = \{A, B, C\}$ y $P = \wp(E)$
- Dar funciones de interpretación para cada no-terminal:

Funciones para $[\![.]\!]_{\langle \text{Elem} \rangle}$ y $[\![.]\!]_{\langle \text{Prog} \rangle}$

$$
\begin{aligned}
[\![a]\!]_{\langle \text{Elem} \rangle} &= \text{A} \\
[\![b]\!]_{\langle \text{Elem} \rangle} &= \text{B} \\
[\![c]\!]_{\langle \text{Elem} \rangle} &= \text{C} \\
[\![(P \,\texttt{union}\, Q)]\!]_{\langle \text{Prog} \rangle} &= [\![P]\!]_{\langle \text{Prog} \rangle} \cup [\![Q]\!]_{\langle \text{Prog} \rangle} \\
[\![(E \,\texttt{into}\, P)]\!]_{\langle \text{Prog} \rangle} &= \{[\![E]\!]_{\langle \text{Elem} \rangle}\} \cup [\![P]\!]_{\langle \text{Prog} \rangle} \\
[\![\texttt{emptyset}]\!]_{\langle \text{Prog} \rangle} &= \emptyset
\end{aligned}
$$

Semántica - Ejemplo (denotacional)

Ejemplo: Sea P el programa

```
((a into emptyset) union emptyset)
```

Entonces

$$[\![P]\!]_{\langle \text{Prog} \rangle} = \{A\}$$

Donde (recordemos):

$$\begin{aligned}
[\![a]\!]_{\langle \text{Elem} \rangle} &= A \\
[\![b]\!]_{\langle \text{Elem} \rangle} &= B \\
[\![c]\!]_{\langle \text{Elem} \rangle} &= C \\
[\![(P \texttt{ union } Q)]\!]_{\langle \text{Prog} \rangle} &= [\![P]\!]_{\langle \text{Prog} \rangle} \cup [\![Q]\!]_{\langle \text{Prog} \rangle} \\
[\![(E \texttt{ into } P)]\!]_{\langle \text{Prog} \rangle} &= \{[\![E]\!]_{\langle \text{Elem} \rangle}\} \cup [\![P]\!]_{\langle \text{Prog} \rangle} \\
[\![\texttt{emptyset}]\!]_{\langle \text{Prog} \rangle} &= \emptyset
\end{aligned}$$

Procesadores de Lenguajes

Dos formas de ejecutar programas:

1. *interpretar*
2. *compilar*

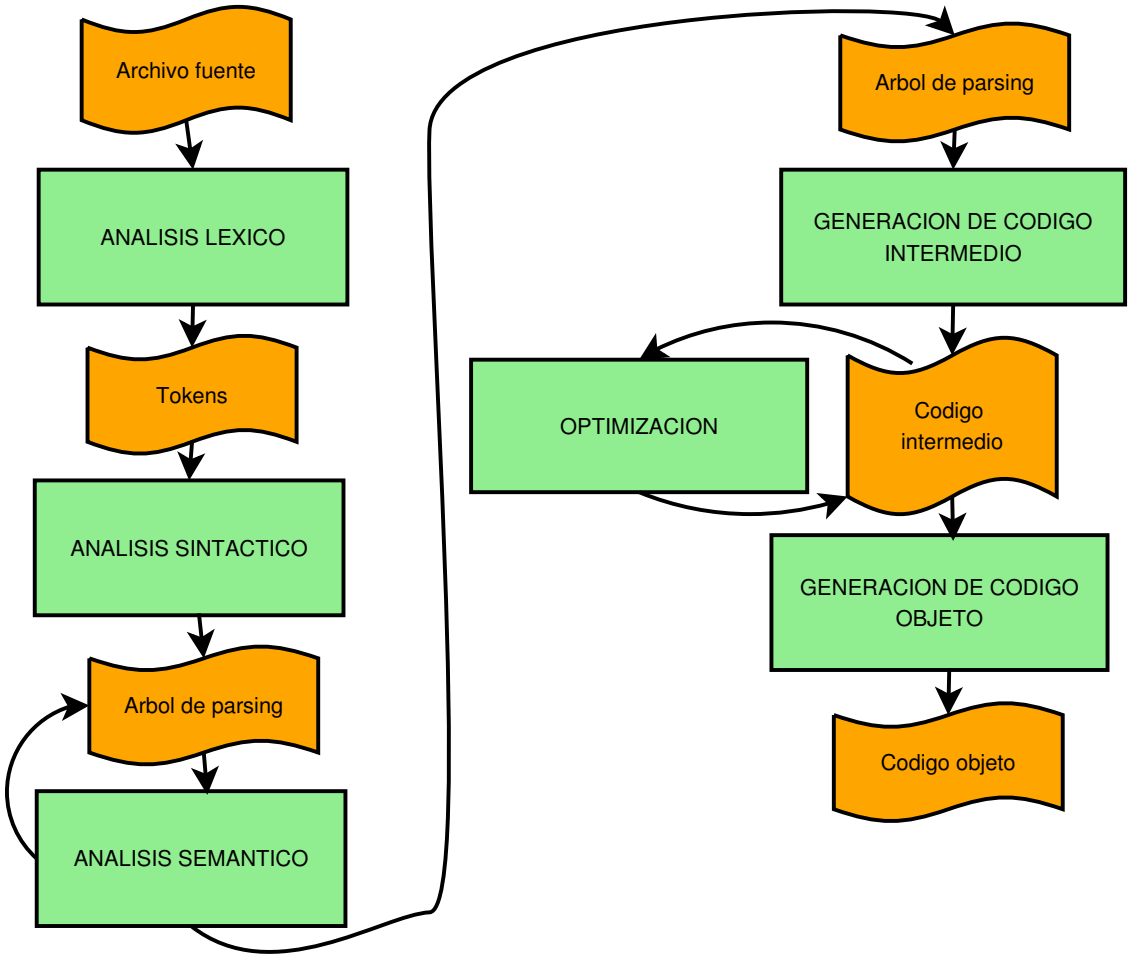
Intérprete

- programa que computa las acciones indicadas por un programa fuente
- es un *meta-programa* - programa que procesa otro programa
- utilizado para prototipar
- utilizado para dar semántica a un lenguaje
- nosotros lo utilizaremos para estudiar conceptos de LP

Compilador

- Traduce un programa *fuente* en un programa *objeto*
- Muchas veces el programa fuente es un LP y el lenguaje objeto un lenguaje de máquina
- También es un *meta-programa*
- Parte de su tarea consiste en verificar que la cadena de entrada efectivamente se corresponde con un programa válido

Compilador


Archivo fuente
ANALISIS LEXICO
Tokens
ANALISIS SINTACTICO
Arbol de parsing
ANALISIS SEMANTICO
Arbol de parsing
GENERACION DE CODIGO INTERMEDIO
OPTIMIZACION
Codigo intermedio
GENERACION DE CODIGO OBJETO
Codigo objeto

Compiladores vs Intérpretes

Ventajas de compiladores

- código compilado ejecuta (mucho) más rápido que código interpretado
- permite procesar módulos por separado

Ventajas de intérpretes

- programa fuente más pequeño que código compilado
- más fácil de escribir y modificar
- provee independencia de plataforma (portabilidad)

Paradigmas de Lenguajes

Diferentes maneras de expresar computación

- imperativo
- funcional
- lógico
- orientado a objetos

Otros: concurrente, dataflow, algebraico, etc.

Distinción a veces no está clara!

Veamos un ejemplo: calcular m^n $(n \geq 0)$

Imperativo

```
result := 1;
while n>0 do
    result := result * m;
    n := n - 1;
end while
```

Evaluación

- computación expresada a través de modificación reiterada de memoria *implícita*
- variables como abstracción de celdas de memoria
- resultados intermedios se almacenan en la memoria
- control basado en iteración

Funcional

```
power m 0 = 1
power m (n+1) = m*power m n
```

Evaluación

- computación expresada a través de la aplicación y composición de funciones
- no hay una memoria implícita
- resultados intermedios (salida de las funciones) son pasados directamente a otras funciones
- control basado en recursión

Lógico

```
power(m,0,1).
power(m,n,result)
  <- minus(n,1,n_sub1),
       power(m,n_sub1,temp_result),
       times(m,temp_result,result).
```

Evaluación

- computación expresada a través de proof search o alternativamente, por definición de predicados recursivos
- no hay memoria implícita
- resultados intermedios son pasados a través de unificación
- control basado en recursión

Orientado a Objetos

```
class Numero
  | val ... |
  instance method
    Valor              Power(pot)
      return val        if pot=0 then return 1
                         else return
                                ((send self Valor) *
                                 (send self Power(pot-1)))
```

Evaluación

- computación a través del intercambio de mensajes e/objetos
- objetos se agrupan en clases, clases de agrupan en jerarquías
- resultados son pasados como parámetros a mensajes

Top five: "Great Works in Programming Languages", B.Pierce

- C.A.R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, 12(10):576-580 and 583, October 1969.
- Peter J. Landin. The next 700 programming languages. Communications of the ACM, 9(3):157-166, March 1966.
- Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 17:348-375, August 1978.
- Gordon Plotkin. Call-by-name, call-by-value, and the λ-calculus. Theoretical Computer Science, 1:125-159, 1975.
- John C. Reynolds. Towards a theory of type structure. In Colloque sur la Programmation, Paris, France, volume 19 of Lecture Notes in Computer Science, pages 408-425. Springer-Verlag, 1974.

Vanguardia

Conferencias Europeas

- The European Joint Conferences on Theory and Practice of Software (ETAPS)
  - Foundations of Software Science and Computation Structures (FOSSACS)
  - Fundamental Approaches to Software Engineering (FASE)
  - European Symposium on Programming (ESOP)
  - International Conference on Compiler Construction (CC)
  - Tools and Algorithms for the Construction and Analysis of Systems (TACAS)
- International Colloquium on Automata, Languages and Programming (ICALP)
- Computer Science Logic

Vanguardia

Conferencias ACM

- Principles of Programming Languages (POPL)
- International Conference on Functional Programming (ICFP)
- Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)
- Programming Language Design and Implementation (PLDI)
- Principles and Practice of Parallel Programming (PPOPP)